J Sign Process Syst (2017) 87:327–341
DOI 10.1007/s11265-016-1204-8

A Customized Many-Core Hardware Acceleration Platform for Short Read Mapping Problems Using Distributed Memory Interface with 3D–Stacked Architecture

Pei Liu[1] · **Ahmed Hemani**[1] · **Kolin Paul**[2] · **Christian Weis**[3] · **Matthias Jung**[3] · **Norbert Wehn**[3]

Received: 15 June 2015 / Revised: 7 September 2016 / Accepted: 8 November 2016 / Published online: 3 December 2016

Abstract Rapidly developing Next Generation Sequencing technologies produce huge amounts of short reads that consisting randomly fragmented DNA base pair strings. Assembling of those short reads poses a challenge on the mapping of reads to a reference genome in terms of both sensitivity and execution time. In this paper, we propose a customized many-core hardware acceleration platform for short read mapping problems based on hash-index method. The processing core is highly customized to suite both 2-hit string matching and banded Smith-Waterman sequence alignment operations, while distributed memory interface with 3D–stacked architecture provides high bandwidth and low access latency for highly customized dataset partitioning and memory access scheduling. Conformal with original BFAST program, our design provides an amazingly 45,012 times speedup over software approach for single-end short reads and 21,102 times for paired-end short reads, while also beats similar single FPGA solution for 1466 times in case of single end reads. Optimized seed generation gives much better sensitivity while the performance boost is still impressive.

Keywords Accelerator architectures · Application specific integrated circuits · Bioinformatics · Computational biology · Coprocessors · Three-dimensional integrated circuits

✉ Pei Liu
peiliu@kth.se

Ahmed Hemani
hemani@kth.se

Kolin Paul
kolin@cse.iitd.ac.in

Christian Weis
weis@eit.uni-kl.de

Matthias Jung
jungm@eit.uni-kl.de

Norbert Wehn
wehn@eit.uni-kl.de

[1] Department of Electronic System, School of ICT, KTH Royal Insitaitute of Technology, 16440 Stockholm, Sweden

[2] Department of Computer Science and Engineering, Indian Institute of Technology Delhi, New Delhi, India

[3] Microelectronic Systems Design Research Group, University of Kaiserslautern, Kaiserslautern, Germany

1 Introduction

DNA sequencing is a technology on detection of the sequence base pairs of nucleotides in a DNA sample. It's the basement of modern life science fields such as medicine, agriculture and forensics. *Next Generation Sequencing* (NGS) technologies have greatly reduced the cost of DNA sequencing and increased data throughput. It randomly breaks down many genome copies of an organism into small pieces, then optically "read" the fragmented string ends in parallel and output as huge amount of randomly fragmented DNA strings with equal length.

Those strings called "*read*" are typically representing 50 ~ 400 *base-pairs* (bp) in length for DNA. In order to reconstruct the original sequence structure of target organism sample, an assembling stage is introduced with two different approaches: de-novo assembly and reference-guided assembly. The de-novo method assembles target short *reads* by comparing each other to create full-length sequences, while reference-guided method assembles target short *reads* against an existed genome model to build a similar but unnecessarily identical sequence. De-novo assemblies are orders of

magnitude slower and more memory intensive than mapping assemblies, meanwhile the latter is much more commonly used in healthcare research of human beings. In this paper, we focus on the reference-guided assembly method, or so called short read mapping.

The mapping process, either indexing the input *reads* and then scanning through the reference genome or aligning each *read* independently against indexed reference sequence, all the approaches within this category follow the same principle: index, search, and local sequence alignment. Rapidly developing NGS technologies produces more and more *read* data than ever before, and the *read*-indexing approaches turned to be deprecated nowadays, thus we focus on database-indexing methods. There are mainly two types of database indexing methods, hash-index and FM-index [1].

The hash-index method leverages the fact that individual genomes differ slightly, so that it is likely that some tiny sub-sequences of a short *read* will match part of the reference genome exactly. These tiny sub-sequences are called *seeds*, hashed value of which shall be evaluated throughout the hashed reference genome to determine the candidate matching locations. For each candidate matching location, corresponding short *read* shall be scored against the reference sequence using fully gapped *Smith-Waterman* sequence alignment [2] for verification. BFAST [3] is an example program based upon this approach.

The FM-index approaches are based on a block sorting data compression algorithm called *Burrows-Wheeler transform* (BWT) [4]. FM-index [1] is a compressed index that combines the properties of the suffix array [5] with the BWT. It allows searching for a pattern using binary search instead of a linear time scan. Although it needs more table accesses than classical hash-index based approaches, it's shown to be one to two orders of magnitude faster in case of limited number of differences between a *read* and the reference genome. Programs within this group performance well on desktop computers, but the running time would be exponential growth with respect to the allowed number of differences. Therefore these tools tend to be less sensitive. Bowtie [6] and BWA [7] are examples based upon this approach.

Modern sequencing machines can generate billions of bp data in very short time. For example, an Illumina HiSeq 2500 sequencing system [8] would generate a maximum dataset of 1000 Gb per single run, which consists of 4 billion *reads* with each length up to 2 × 125 bp. Thus the data processing of these *reads*, traditionally performed by software and general purpose computers, is becoming the bottle-neck of the sequencing industry. A number of acceleration platforms have been proposed based on FPGAs [9, 10] and GPGPU [11]. Both FM-index and hash-index programs have been significantly accelerated, but a series of restrictions are applied among those implementations. Performance of FM-index programs is greatly reduced in case of high difference rate even with hardware acceleration. Hash-index short read mapping can achieve high accuracy with high difference rates, but hardware acceleration is restricted by memory sub-system in terms of access latency and memory size [9].

In this paper, we turn to architectural innovation to address the performance bottleneck caused by memory sub-system in traditional FPGA designs. We present a novel architecture for short read mapping problem of hash-index method, using customized many-core and 3D–stacked technologies.

The salient contributions of this paper are:

- We present an analysis of hash-index based short read mapping problem. We summarize the application with computation stages and data flow, and identify the kernel functions which consume a dominant portion of the overall execution time.
- We describe a processing core with dedicated DRAM channels, which is designed to facilitate both 2-hit string matching and *Smith-Waterman* local sequence alignment operations concurrently. Dataset partitioning, customized interconnections and 3D–stacked technologies enable the use of customized many-core architecture for massively parallelization.
- We introduce a memory access policy to maximize the throughput of local DRAM access while devoid the bus congestion in a circuit-switching many-core system. This policy has been adopted to provide both high bandwidth and low access latency.
- We present results from prototyping stripped down implementation of our design on a commodity FPGA, representing the performance of a single processing core of our architecture. Then we scale the results to give a plausible estimation of our design with customized many-core architecture and 3D–stacked DRAM. Based on conservative area estimation and simulation results, we show that our design achieves the highest performance and greatest energy efficiency comparing with all other known accelerators.

The remainder of this paper is organized as follows. Section 2 introduces hash-index based short read mapping, *Smith-Waterman* local sequence alignment, and 3D–stacked DRAM technologies. Related works that accelerating hash-index based short read mapping with FPGAs are also briefly covered. Section 3 describes multiple optimizations that have been applied in our platform to address both performance and sensitivity improvements, while Section 4 introduces detailed implementation of our customized many-core architecture with 3D–stacked DRAM. Experimental results are presented in Section 5. Our architecture (with its concomitant software support) is the first reported solution on accelerating short read mapping problem of hash-index method with 3D–stacked VLSI architecture. Finally Section 7 presents conclusions.

2 Background and Related Work

2.1 Hash-Index Based Short Read Mapping

Hash-index based short read mapping is a deterministic method. During data preparation stage, a large index of the reference genome is compiled, which maps every *seed* resides in the reference genome to the *candidate alignment location* (CAL) where it resides. This index is formed as an array of *seed-CAL* pairs and sorted by *seed*, creating a structured table called *CAL table*.

The complete *CAL table* is extremely large in real practice, thus a *pointer table* is built to help accessing the specific range of the *CAL table* without traverse it completely every time. The *pointer table* is indexed using prefix portion of each *seed*. Each entry of the *pointer table* contains an address to the *CAL table* along with the size of the data bucket. Each CAL bucket contains a series of locations where the actual *seed* resides in the reference genome, as well as the hashed rest nucleotides of those *seeds*. All *seeds* in the same CAL bucket share the same address, but they may differ by their least significant nucleotides, called *key*.

After data preparation, the general idea of hash-index based short read mapping algorithm is shown in pseudo code as algorithm 1:

Algorithm 1 Original hash-index based mapping algorithm

```
1: pre-generate reference genome for index tables
2: for i = 1 to num_reads
3:    read_curr = reads[i]
4:    seeds, num_seeds = Get_seeds(read_curr)
5:    for j = 1 to num_seeds
6:       seed _curr = seeds[j]
7:       addr_seed, key_seed = Hash(seed_curr)
8:       bucket_cal, num_cal = ptr_bucket[addr_seed]
9:       for k = 1 to num_cal
10:         cal_curr, key_ref = bucket_cal [k]
11:         if key_seed = key_ref
12:            ref_curr = seq_ref[cal_curr]
13:            score[k]= S-W (read_curr, ref_curr)
14:         end if
15:      end k
16:   end j
17:   report Max{score}
18: end i
```

For current short read *curr_read*, all the possible *seeds* are extracted with a defined length between 18 to 22 character symbols accordingly and then hashed to get an address-key pair. The prefix part of a hashed *seed*, *addr_seed*, with 14 to 16 nucleotides in length is used as an address to query the *pointer table*, and the significant suffix part is reserved as *key_seed*. A valid hit to pointer table using *addr_seed* retrieves the beginning address *bucket_cal* and the size of relevant data bucket *num_cal* in the *CAL table*, which consists of a number of *CALs*.

For each record of the data bucket retrieved from *CAL table*, the hashed candidate alignment location from reference genome, *key_ref*, is compare with the suffix of hashed *seed key_seed*, and a hit is reported along with the location to reference genome, *cal_curr*.

If any CAL matches the current *seed* indicated by 2-hit string matching, *Smith-Waterman* local sequence alignment shall be applied between the current short *read* and reference genome sequence at each of associated CALs pointed by corresponding *cal_curr*. The location with the highest matching score is chosen as the alignment location of the current short *read*.

An example demonstrating the complete *seed* matching flow is shown as Fig. 1.

Therefore the hash-index based short *read* mapping can be summarized into 3 stages:

- For each *read*, *k-mers* are obtained called *seed* string and hashed.
- Prefix and suffix of hashed *seed* string are used for 2-hit string matching against hashed reference sequences.
- *Smith-Waterman* local sequence alignment is executed between the short *read* and reference sequence at each substring matched sites for verification.

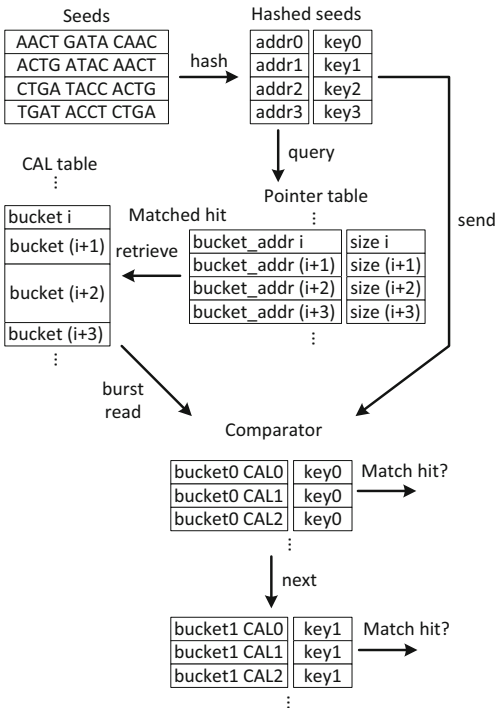


Figure 1 *Seed* matching flow.

Similar to most scientific programs, a dominant portion of overall execution time is consumed by a few stages of all the applications of interest in hash-index based short read mapping. The profiling of BFAST presents that 48% of overall run time is consumed by 2-hit string matching and another 40% by *Smith-Waterman* local sequence alignment on our software benchmark platform.

This flow is sequentially executed in software but also well-suited for massively parallelism acceleration, as operations regarding each *seed* as well as each *read* are independent from others, thus the problem can be partitioned into small sub-instances. A more detailed description of hash-index based short read mapping can be found in BFAST [3].

2.2 Smith-Waterman Local Sequence Alignment

The pairwise alignment is the most fundamental class of method in the field of biological sequence analysis; a large number of complex applications have incorporated the pairwise alignment algorithms within their processing flow. *Smith-Waterman* sequence alignment [2] is the standard model for local sequence alignment, that only the sub-sequences which present the highest similarity are relatively aligned between two sequences.

For sequence a and b with length m and n respectively, adopting *Smith-Waterman* alignment would build a distance score matrix filled with aligned scores at first, where each element is the optimal local alignment score representing the evolution distance between the segments in two sequences. For *Smith-Waterman* local sequence alignment using affine gap, the distance scores between sequence a and b are assigned recursively to fill the distance score matrix as shown in Algorithm 2, where $S_{i,j}$ is the substitution matrix score or entry of residues i and j, d is gap open and e is gap extension.

Algorithm 2 Distance matrix filling stage of Smith-Waterman local sequence alignment (Affine Gap)

1: for $i = 1$ to H, $j = 1$ to L
2: $M_{i,0} = 0$, $M_{0,j} = 0$ // Initializing
3: end i, j
4: for $i = 1$ to H , $j = 1$ to L
5: $M_{i,j} = \max\{\{\{ M_{i-1,j-1},\ I_{i-1,j-1},\ D_{i-1,j-1}\} + S_{i,j} \}, 0\}$//Match
6: $I_{i,j} = \max\{ M_{i-1,j} - d,\ I_{i-1,j} - e \}$ // Insertion
7: $D_{i,j} = \max\{ M_{i,j-1} - d,\ D_{i,j-1} - e \}$ // Deletion
8: $F_{i,j} = \max\{ M_{i,j},\ I_{i,j},\ D_{i,j} \}$ // Filling
9: end i, j

After generation of distance matrix, *Smith-Waterman* alignment adopts a *backtracking* stage to find the final optimal alignment. The starting point of best local alignment chain is given by the maximum distance score as (1), and then tracking back diagonally till "0".

$$\mathrm{Start}_{\mathrm{local}} = \max_{i,j=1}^{m,n} \{M_{ij}\} \tag{1}$$

2.3 Related Works

BFAST [3] by Homer et al. is a short read alignment tool for large scale genome resequencing using hash-index method. In general case, the first 18 to 22 nucleotides of each *read* are used as a *seed* and hashed for 2-hit string matching, and the matching results are stored into a compressed file. Then the program use Smith-Waterman local sequence alignment to verify each candidate matching site sequentially, and report the final mapping results.

Olson et al. [9] first proposed a FPGA based solution to accelerate modified BFAST algorithm. They hashed the *seeds* using a 1–1 hash function on the all 44-bits, and then use the most significant 30 bits of the result to address the *pointer table*. The remaining 14-bits are stored as *key* part to identify the *seed* amongst the *CALs* in the *CAL* table. Their platform consists of multiple FPGA modules, thus they have to partition the *CAL table* and *pointer table* into small blocks, so that the system has the ability to compute large genome dataset in real. They reported a 250× speedup versus the original BFAST software with 91% sensitivity, and 31 times speedup versus Bowtie [6] with 80% of short reads aligned.

Sogabe et al. [10] also proposed a FPGA based platform to accelerate hash-index based BFAST, while they focused on the hash matching functions but not the complete BFAST application. They modified the BFAST source code for shorter length of address pointer, which resulted to a longer key part for more precise matching. Allowing substitutions in the key filtering process, the search becomes more tolerant against substitutions. As a result, their system reported to be more sensitive than BFAST, Bowtie and implementation by Olson et al. With a very high matching rate of 99.48%, they retain a speedup of 2.5 times versus Bowtie.

There has been no reported GPGPU implementation of BFAST, since the hardware architecture of which is weak on frequent random memory access with tiny payloads.

2.4 Random Memory Access

Figure 2 demonstrates benchmark results for memory bandwidth utilization generated by *Randmem* [12] on our i7–960 platform, which has a theoretical maximum memory bandwidth of 25.6 GB/s (DDR3–1066 in 3 channels) [13, 14]. For various size of data blocks stored in DDR3 memory, randomly reading from or writing to one 64bit double/integer value is normalized as real memory bandwidth, which

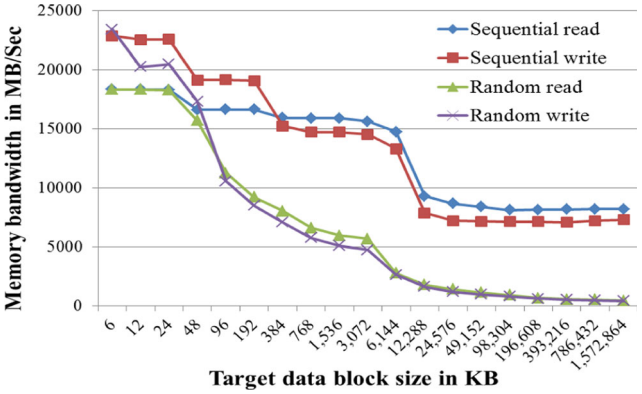


Figure 2 Memory bandwidth benchmark on i7-960.

dropped to 470 ~ 1825 MB/s in case of target data blocks larger than on-chip cache size (8 MB on i7–960 processor). There has been no reported GPGPU implementation of BFAST, since the hardware architecture of which is weak on frequent random memory access with tiny payloads.

The principles of random read and write in *Randmem* benchmark is similar with the case of 2-hit string matching algorithm in BFAST. During the string matching, various tables stored in main buffer are randomly retrieved with tiny payload. It could hardly benefit from memory bursts, as the payload is very small on each access and the access address is unpredictable. This principle leads to extremely low memory bandwidth utilization in real, which is the major cause of the sharp drop over random access curves in Fig. 2.

FPGA and GPGPU also suffer from this bottleneck, since each (G)DDR2/3/4/5 SDRAM channel could only response to limited request at the same time. Therefore theoretical bandwidth does not yield high random access bandwidth in such extreme application, which results relatively poor performance on FPGA implementations [9, 10].

2.5 MPSoC and 3D–Stacked Technologies

Interconnect scaling has become one of the most crucial challenges in chip design, and is expected to get worse in the future. *Multiprocessor System-on-Chip* (MPSoC), *Network-on-a-Chip* (NoC) and 3D stacking technologies are expected to be the promising solutions to overcome many of these challenges.

For hardware acceleration, the demand of memory bandwidth is exponentially growing along with increasing compute ability, which becomes a bottle-neck of massively paralleled systems. The idea of distributing memory controllers to manage multiple DRAM channels has been proposed as a solution. A novel scheme for memory controller placement in many-core processor was presented by Abts et al. [15], while they retain a 2D mesh of chip layout.

This MPSoC 2D array is capable for solving BFAST problem in theory, but its performance is restricted by external memory bandwidth. For 2-hit string matching algorithm, most of the computations are finished in one clock cycle and no further propagation on data-path. Smith-Waterman local sequence alignments are rarely executed due to small table partition. Therefore there is no efficient way to make use of most of the processing resources in an MPSoC 2D array to accelerate BFAST application, since the dominant external DRAM accesses are random read, and the available number of on-chip memory controllers is constrained by limited PCB layout resources.

Loi et al. [16] first proposed distributed memory controller architecture for vertically stacked memory over a many-core logic die, which ensures ultra-low latency access to the memory modules on top of each processing element. On the horizontal logic plane, they connected logic cores through NoC, therefore the vertically stacked memory system can be modeled with the abstraction of memory neighborhood, as processing elements within an array have fast access and high bandwidth to a vertical stack of memory banks on top of it. Communication to these local modules do not travel through the NoC, thus it takes full advantage of the lower latency on vertical TSV interconnect, resulted to a significantly speedup in case of local accesses. Experimental results demonstrate significant bandwidth improvement that ranges from 1.44× to 7.40× compared to the JEDEC standard, with peaks of 4.53GB/s for direct memory access.

Recently Weis et al. [17] explored the 3D–DRAM architecture design space. They proposed a highly energy-efficient DRAM subsystem for next-generation 3D–integrated *System-on-a-Chip* (SoC), consisting of a 3D–DRAM controller and an attached 3D–DRAM cube with fine-grained access and a flexible (WIDE-IO) interface. They also investigate different DRAM families and densities from 256 Mb to 4Gb per channel, as well as the energy efficiency issues. The implementation results of the proposed 3D–DRAM subsystem show that energy optimized accesses to the 3D–DRAM enable up to 50% energy savings compared to standard accesses.

Micron has shipped the example devices of Hybrid Memory Cube (HMC) products [18], which could provide a raw link bandwidth of 320Gb/s. As a commercial product in early evaluation stage, most key technical specifications are being kept as secret. The sequential link transactions consists of data in assembled packets, but the most recent HMC specification v1.1 [19] doesn't provide the detail about the packet structure, as well as the clock latency of TX/RX, thus the utilization ratio of HMC data link is a mysterious. We can hardly make rough performance estimations for HMC in real application at this moment.

3 System Optimizations

In order to improve the performance of hash-index based short read mapping problem, a series of optimizations has been implemented and introduced in this section.

The original software implementation has been optimized into a HW-SW co-design architecture, which makes use of hardware massively parallelization to accelerate multiple execution stages. In order to fulfill the bandwidth requirement of this massively parallelized architecture, a 3D–stacked memory sub-system has been proposed with optimized access policy.

3.1 Data and Task Partition of Hash-Index Based Short Read Mapping

According to Fig. 1, the *seed* matching flow of hash-index based short read mapping is sequentially executed in software but also well-suited for massively parallelism acceleration, as operations regarding each *seed* as well as each *read* are independent from others, thus the specific task can be partitioned into small sub-instances. In order to parallel such a matching task, we should partition the *CAL table* and *pointer table* into small sub-tables which include different data regions respectively, thus different *reads* can be processed on multiple sub-instances.

To partition the tables with n ways, both the *CAL table* and *pointer table* are partitioned by the first $log_2(n)$ bits of the *seed* hash, thus *seeds* can be searched against different data regions with n instances in parallel. Additionally, as $log_2(n)$ bits have been used to address the sub-instances, the address bits in pointer table can be reduced by $log_2(n)$ bits.

On the other hand, all the matched CALs shall be evaluated with *Smith-Waterman* local sequence alignment. Each matched CAL is used to retrieve corresponding reference genome clip stored in memory, which is to be aligned against the short *read* that contains the matched *seed*. This procedure can be also partitioned for parallelization, since each run of *Smith-Waterman* alignment is executed independently between a specific short *read* and a clip of reference genome. The final alignment score will be temporary stored and then reported back to host processor for further processing.

Our partitioned hash-index based short read mapping for hardware acceleration in parallel is shown in pseudo code as algorithm 3. *Seeds* within each *read* are extracted and hashed by dedicated building block and stored in an individual buffer, then distributed to different processing cores for 2-hit string matching in parallel. Operations between line 14 and 25 are partitioned 2-hit string matching task to be executed in parallel, while lines between 27 and 32 are also accelerated with customized *Smith-Waterman* processing element array, which is to be introduced in next sub-section.

Algorithm 3 Partitioned hash-index mapping algorithm

```
 1: pre-generate reference genome for index tables
 2: partition CAL and pointer table with num_parts
 3: divide reference genome sequences into num_parts
 4: for i = 1 to num_parts
 5:     transfer data from host to processing core i
 6: end i
 7: for j = 1 to num_reads
 8:     read_curr, id_read = reads[j]
 9:     seeds, num_seeds = Get_seeds(read_curr)
10:     for k = 1 to num_seeds
11:         addr_seed[k], key_seed[k]  = Hash(seeds[k])
12:     end k
13:     seed_buffer= {addr_seed, key_seed}
14:     for l = 1 to num_parts
15:         addr_seed[l], key_seed[l] = seed_buffer[j,l]
16:         bucket_cal, num_cal = ptr_bucket[addr_seed[l]]
17:         for m = 1 to num_cal
18:             cal_curr, key_ref = bucket_cal[m]
19:             if key_seed = key_ref
21:                 cal_match = cal_curr, id_match = id_read
22:                 broadcast(cal_match, id_match)
23:             end if
24:         end m
25:     end l
26: end j
27: for n = 1 to num_parts
28:     sync(cal_match, reads[id_match])
29:     ref_match= refseq_local[cal_match]
30:     score[id_match] = S-W (reads[id_match], ref_curr)
31:     Save(score[id_match])
32: end n
33: for each id_match
34:     report Max{score[id_match]}
```

3.2 Banded Smith-Waterman Local Sequence Alignment

The distance score matrix filling of *Smith-Waterman* local sequence alignment is defined to be the path from the top-left corner to the bottom-right corner of the score matrix. Therefore a *wave-front* method is commonly used for parallelization by adopting *systolic array* [20], which reduces the time complexity of complete filling from $O(m \times n)$ to $O(m \times n/p)$, where m and n is the length of the sequence pair to be aligned, p is the number of processing elements in parallel. The *wave-front* distance score matrix filling and afterwards *backtracking* flow is shown as Fig. 3:

The *wave-front* method is a promising way to accelerate distance score matrix filling stage in parallel, which can be achieved by using a *systolic array* with depth of *m processing*

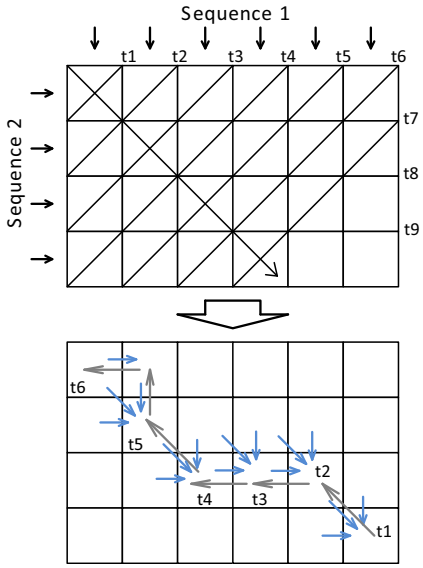


Figure 3 Workflow of *Smith-Waterman* local sequence alignment.

elements (PE). Assume that n instances of *systolic array* are adopted in another order of parallelization, the initial genome sequence input can be encoded using 2 bits to represent DNA sequence symbols, and the short *read* input does not rely on memory access, thus n is theoretically given by (2):

$$n = \frac{B_{eff\_in}}{f_{\text{array}} \times 2} \tag{2}$$

Where B_{eff_in} is the effective external dynamic input bandwidth and f_{array} is the clock frequency of systolic array. PEs in the deeper stages of array could receive data spread by former stages, while the output of each PE is stored to fill the distance matrix, which would produce w bits of data to be stored in each computation cycle, where w is 21 and denotes the output bit-width of a PE. In this case, the theoretical output bandwidth of our systolic array B_{array_out} without buffer is given by (3):

$$B_{\text{array\_out}} = m \times n \times f_{\text{array}} \times 21 \tag{3}$$

Assume $B_{eff\_in} = B_{array\_out}$, which is roughly the real case of sequential operations over external memory sub-system, then m = 2/21 from (2) to (3). This mismatch makes classical approach of distance matrix filling benefit less than anticipated on massively paralleled accelerators, since all the distance scores shall be saved for backtracking stage.

By observing the hash-index based short read mapping, we found out that the accurate local alignment between short *read* and reference genome is not going to be reported for each *seed* but only the one with maximum alignment score. With *seed* length of 22 and *read* length of 100 ~ 400, the number of *seeds* to be evaluate can be extreme, thus we turn to an alternative method that would skip backtracking stage, called ***banded*** *Smith-Waterman* alignment [21] and present in pseudo code as algorithm 4:

Algorithm 4 Banded Smith-Waterman local sequence alignment without storing filling (Affine Gap)

```
band_gap = k, band_chk = 0
for i = 1 to H, j = 1 to L
      M_{i,0} = 0, M_{0,j} = – d – (i – 1) × e        // Initializing
end i, j
for i = 1 to H , j = 1 to L
   M_{i,j} = max{{ M_{i-1,j-1}, I_{i-1,j-1}, D_{i-1,j-1}} + S_{i,j} }, 0}//Match
   I_{i,j} = max{ M_{i-1,j} – d, I_{i-1,j} – e }          // Insertion
   D_{i,j} = max{ M_{i,j-1} – d, D_{i,j-1} – e }          // Deletion
   F_{i,j} = max{ M_{i,j}, I_{i,j}, D_{i,j} }             // Filling
      if F_{i,j} = I_{i,j}
         band_chk = band_chk – 1
      else if F_{i,j} = D_{i,j}
         band_chk = band_chk + 1
   if band_chk = band_gap
      stop, report band_gapped_unmatch
end i, j
report band_gapped_match
```

The yellow lines in Fig. 4 indicate *wave-front* computation with a threshold band gap of 3; blue, orange and green lines indicates match, insertion and deletion event respectively. It leverages the fact that correctly mapped genomes differ slightly, therefore alignments with too many insertions or deletions resulting low alignment scores are stopped right on reaching the band gap. The longer matching chain gives higher accumulated alignment score, which means higher possibility to be

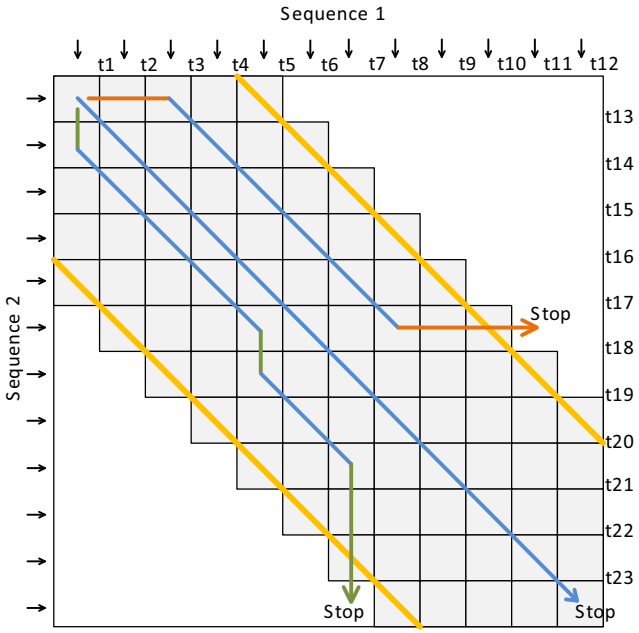


Figure 4 Banded *Smith-Waterman* local sequence alignment.

correct mapping. Therefore the *Smith-Waterman* local sequence alignment between a short *read* and reference genome can be optimized into three stages:

- For each *CAL*, distance scores are generated and accumulated with banded gap. Alignment stops whenever reaching the band gap or all sequence elements have been aligned. No distance scores will be stored in this stage.
- For each *CAL*, accumulated alignment scores of each *CAL* are compared to find out the maximum one.
- The *CAL* with maximum alignment score is reported as the corresponding read mapping with quality information. Formal *Smith-Waterman* alignment can be executed on that site if interested at exact alignment result.

This approach boost the *Smith-Waterman* local sequence alignment stage by an order of magnitude as the *wave-front* computation without matrix filling rely on input bandwidth of systolic array only.

3.3 3D–Stacked DRAM with Scatter/Gather Read Policy

According to JEDEC compliant DRAM model, a *write* request is executed in 3 phases: the target *row* and *bank* is activated during the *activate* phase, and then the column is selected while the data is written in the write buffer during the *write* phase. The *precharge* phase is needed only if the row must be deactivated; as the *precharge* command writes back the data in this row, then this bank is ready for a new request in a different row. The read request is slightly different: there is a latency of several clock cycles before the data reaches the I/O during the second phase.

For 3D compliant DRAM model proposed by Loi et al. [16], *Activate* and *Precharge* phases show the same latency as JEDEC since they rely on the physical interconnections, while the *Read/Write* phase is shorter due to improvement on the I/O interface and lean column decoder architecture. The protocol translation between processor and DRAM is still needed but the access time is much less than classical approaches. Our design follows this model with further optimizations over read access.

We assume that all banks in a DRAM channel share a common 64-bit wide TSV data bus, a single channel is achieved by 8 banks in 8 layers, each bank with a size of 64 Mb. While 64 bits seems to be outdated, 128 bits or more does not give better performance in this specific case. For 2-hit string matching, over 80% of DRAM accesses are random read of 64 bits, and the rest are also random read but with length 8. For Smith-Waterman local sequence alignment, the 64 bit read with random address would last for length 8 at the most. Therefore wider DRAM channel does not help a lot but consumes a lot of extra energy and silicon spaces, which is not acceptable.

Wide I/O DRAM has been standardized by JEDEC (JESD229) [22] as a 3D DRAM solution for embedded SoC systems with lots of low power consideration. In order to conform to the WIDE I/O standard, we assume using LVCMOS signaling for interconnection so that we could avoid using complex SSTL interface, which results in both silicon and power efficiency. We adopt the *wafer-to-wafer* bumpless stacking process which provides aggressively small TSVs at an extremely fine pitch (5 μm). This approach provides much better silicon efficiency than micro-bump based die-to-die stacking with large pitch (20 ~ 40 μm).

The *Scatter/Gather* read policy has been adopted in the memory controller for each DRAM channel, thus the DRAM *read* is optimized for random or strided accesses in 64bit, instead of the classical cache line access policy. This customized policy allows processing cores to directly address individual words in DRAM, which dramatically increased effective memory bandwidth of random read as Fig. 5.

Figure 4 demonstrates the differences of read access between classical 2D DRAM and our 3D DRAM, where the small blocks of active rows indicate data read. For 2D DRAM rank, the data is distributed over all banks, and each access targets a small fraction of the same active address to all banks. *Scatter/Gather* read policy in our 3D architecture allows accessing different banks at the same time, while the target address can be different between each bank. Although data from different banks can't be transferred altogether, the I/O width to each bank is exact the same as the channel width, therefore random read from different banks can be sequentially accomplished in continuous clock cycle, so that achieving much higher bandwidth efficiency.

Data-path latency is another factor that affects DRAM random access, as cache line won't be of much help in this case. Signal from classical 2D DRAM has to be transferred through PCB, SSTL pads, memory controller, and system bus before reaching PE, while in our 3D architecture it becomes much simpler and more efficient. We don't even have to consider the system bus latency in general 3D designs, as remote access to local DRAM channels is avoided by careful dataset partitioning and task scheduling.

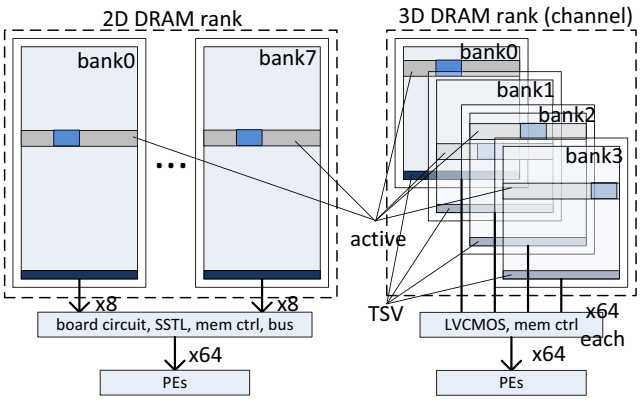


Figure 5 Read accesses over classical 2D DRAM and our 3D DRAM.

Figure 6a shows the layout structure of a single DRAM core tile for 3D architecture, which forms the basis to compose 3D DRAM layers. The tile size is 64 Mb, and we assume 64 I/Os per tile (IOPT) and a page size of 1 kB. Figure 6b depicts a high-level view of our 3D integrated architecture over single stacked tile. Note that the two stack of DRAM memory tiles are accessed using an alternative DRAM interfacing protocol individually.

Our design does not rely on NoC since the partitioned *seed* matching as well as *Smith-Waterman* algorithm takes use of local data only through dataset partitioning. Although our design is many-core architecture on the horizontal logic plane, there is no remote access between local DRAMs and neighborhood processing cores. Each processing core works independently with two integrated memory controllers exclusively, which interfaces with a 3D–stacked DRAM of 64 MB in size respectively. Partitioned *CAL table* is stored in one of the DRAM stack, as well as *partitioned pointer* table and reference genomes in another. This approach provides an extremely high bandwidth for fast local access without consideration of remote access during computation. In case of write or read from host processor, each local processing core will be accessed in a serialized queue, thus no congestion shall be considered on shared bus. Therefore the scalability of our architecture is not restricted by data-path or problem size, but relies overall power dissipation and manufacture cost.

4 Implementation

4.1 Processing Core

The operations within a processing core mainly have two data-paths: one for 2-hit string matching and another for *Smith-Waterman* alignment. The block diagram for single processing core corresponding to the paralleled short read mapping with hash-index method is shown as Fig. 7:

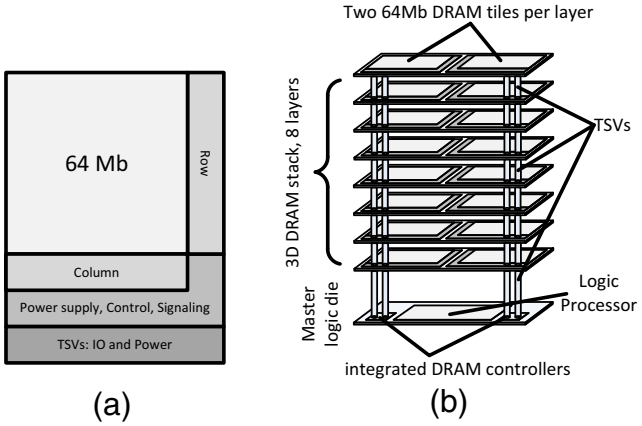


Figure 6 **a** 3D DRAM core tile of 64 Mb **b** 3D architecture of single processing core stack.

A hashed *seed* is input as a combination of *addr_seed* and *key_seed* part. Each *seed* consists of 22 nucleotides, thus a hashed *seed* is represented by 44-bits. The *addr_seed* could be varies between 24 to 32 bit, and *key_seed* could be 20 to 12 bit respectively.

A hashed *seed* is input along with a *read id,* indicates which *read* that this *seed* belongs to, although the process of *read id* is omitted in Fig. 7. The *addr_seed* is used to search the partitioned *pointer table* stored in DRAM1, which forms the first hit of 2-hit string matching. If current *addr_seed* is a valid address to the partitioned *pointer table*, corresponding *CAL bucket* will be read out from DRAM2. Then the *keys* stored in that *CAL bucket* will be compared with *keey_seed* respectively. A 2-hit string matching event will be broadcasted to all the processing cores through shared bus along with matched *CAL* and the *read id*. This stage corresponds to line 7 to line 26 of algorithm 3.

On the other hand, all the broadcasted matches shall be evaluated with *Smith-Waterman* alignment. Each matched *CAL* is used to retrieve corresponding reference genome clip stored in DRAM1, which is to be aligned with the short *read* that contains the matched *seed*. The final alignment score will not be stored but only report whether the banded aliment is completed for the whole *read* or not. Line 27 to 32 of our paralleled algorithm represents this stage, which is detailed with algorithm 4.

4.2 Customized Many-Core Architecture

Our access scheme for short read mapping with hash-index method is proposed as Fig. 8, where local accesses to memory interface are routed directly to the DRAM controllers, while remote accesses are routed to circuit switching bus.

For local access, the transfer takes the advantage of the low latency and high bandwidth of direct access; therefore an entire cache line (or simply a single word) can be written or read in a few clock cycles. A remote request is only considered from external host processor but not neighborhood processors, while the partitioned *CAL table*, *pointer table* and reference genome sequence are written from host processor through burst access, or the short *reads* with aligned scores are read back.

The data flow of our accelerator is demonstrated as Fig. 9. There are three individual broadcast buses colored in gray,

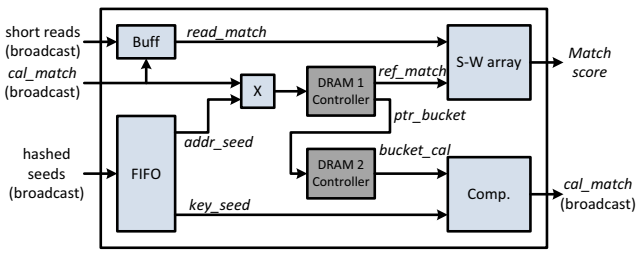


Figure 7 Processing core of acceleration platform for hash index mapping.

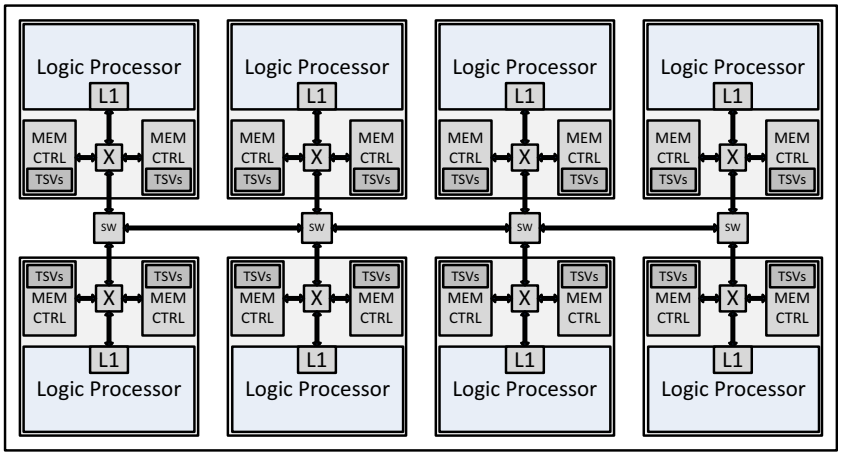


Figure 8 Access scheme of processing core array in 2D mesh.

blue and brown respectively, which are the shared data input of short *reads*, hashed *seeds* and matched *CALs* for all the processing cores. Each processing core has a control unit to determine whether to accept or bypass those data input due to the data range of partitioned dataset stored in local DRAMs. This control unit is omitted in Fig. 9 to not clutter it.

Alignment buffer will collect all the alignment results from Smith-Water local alignment and transfer back to host on the fly. For paired end *reads*, each pair will be evaluated for *seed* matching, and then *Smith-Waterman* local alignment will be applied if any single sequence has matched CAL.

There are 10 different *seeds* from each short *read*, corresponding to 10 different pre-generated *CAL index tables*. The hash index mapping computation is executed sequentially for each CAL index tables due to the requirement of data storage space. In order to avoid redundant computation, mapped *reads* which passed banded *Smith-Waterman* alignment will be recorded and skipped for the rest computations. This approach would avoid more than 80% of computation in the rest iterations.

4.3 Smith-Waterman Processing Element Array

The short *read* shall be aligned to the reference genome according to each *CAL* using alternative banded *Smith-Waterman* local alignment, as it's globally aligned against a local section of the reference. Thus we will receive the alignment of the entire short *read* to a subsequence of the reference genome.

When using the affine gap model, d is a positive value represents the gap open penalty, while e is a positive value

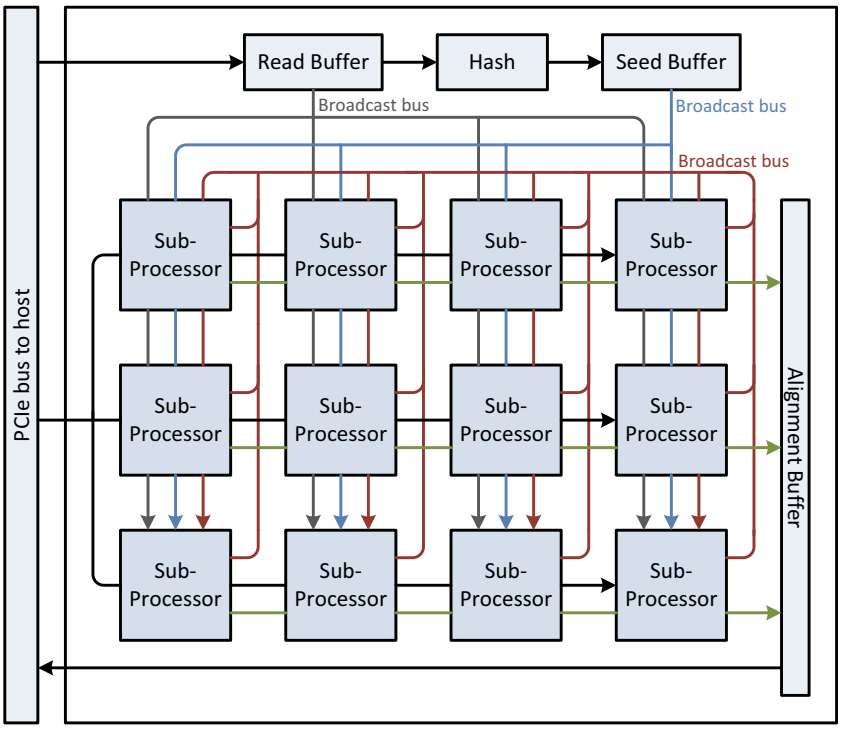


Figure 9 Data flow of the accelerator in 2D mesh.

represents the gap extension penalty. The matrix shall be computed by sweeping *i* from 1 to the length of the short *read*, and by sweeping *j* from 1 to the length of the subsequence of reference genome. In our design, initial gap values shall not be all 0 used in classic Smith-Waterman algorithm. According to the globally alignment of the short *read* to the subsequence reference genome, the scoring matrices are initialized with:

$$M_{0,j} = -d-(i-1) \times e \tag{4}$$

With the help of affine gap, long insertion or deletion chains scores are verified to be more biologically accurate than short insertion/deletion by linear gap. The cost to begin an insertion or deletion chain is higher than the cost to continue a chain in this case. Banded *Smith-Waterman* alignment would just stop work in case of small gap, which would reduce overall sensitivity. Therefor those short *reads* without positive mapping found by banded *Smith-Waterman* alignment shall be aligned with regular Smith-Waterman again. This does not leads to high performance impact since such *reads* is very few in count. A well optimized band gap will be helpful as well.

Acceleration of Smith-Waterman algorithm on FPGA is normally implemented as a *systolic array*. In case of banded *Smith-Waterman* local alignment, the length of the systolic array can be very short considering the narrow band gap, while it also can be much longer since we would like to retain the ability to accelerate regular *Smith-Waterman* alignment. The length of short *reads* and reference genome is neither very long nor very short, and the symbol length of which varies between 50 to 400 in most case. Therefore we use a reconfigurable systolic array that can be reconfigured into multiple individual instances. The computation can be still paralleled, demonstrated as Fig. 10:

4.4 3D–Stacked Architecture

Our customized many-core architecture works in classical ASIC design space, but its scalability is strictly limited by the physical availability of DRAMs. If equipped with 4 DRAM channels per processor chip, similar to the existing commercial products, we can have only two sub-processors on single chip. 3D–stacking technologies eliminate this limitation and we can enhance memory parallelism with novel memory architecture. An overview of our 3D–stacked architecture is illustrated as Fig. 11 for single processing core:

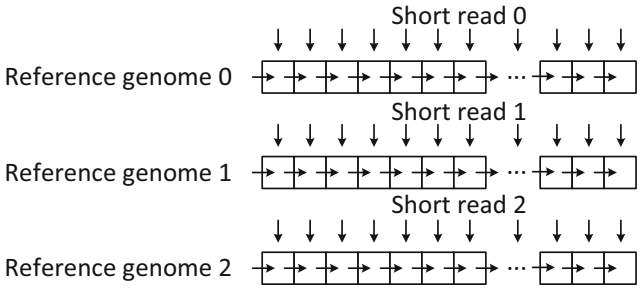


Figure 10 Multiple systolic array instances in parallel.

In Fig. 11, each DRAM layer consists of two DRAM tiles. Each tile is a DRAM unit of 64 Mb; each processing core has two DRAM stacks; there are 4 × 8 = 32 processing cores; the DRAM stacks are formed with 8 layers; thus we have an on-chip storage space of (64 × 2 × 32 × 8) / 8 = 4096 MB in total.

According to [17], a typical 3D DRAM core tile of 64 Mb occupies 1.92 mm^2 in 45 nm tech node, thus the chip area we planned is approximately 130 mm^2.

5 Experimental Results

In this section, we evaluate and compare the overall performance and silicon efficiency of our platform with software and scaled FPGA platforms.

5.1 Evaluation Datasets and Platforms

To assess the performance on real data, we downloaded a dataset which consists of about 63.9 million pairs of 101 bp *reads* from DNANEXUS (ncbi run SRR867061). These *reads* were produced by Illumina HiSeq2500 for NA12878, a female human included in the 1000 Genomes Project [23]. Both single and paired end *reads* from this dataset were used for this performance evaluation. *Reads* were mapped to the human genome issue hg38 from UCSC [24].

The baseline for performance evaluation is a workstation equipped with Intel Core i7–960 3.20GHz CPU, 48 GB RAM and Redhat Enterprise Linux v5.8 × 64 running on it. Software performance corresponding to the baseline is reported as "BFAST" for BFAST v0.7.0a with all the eight CPU threads function in parallel using "-n 8" switch, other settings remain default. Source codes are compiled with *gcc* and "*-O3*" optimization flag.

Our VLSI design is based on TSMC 45 nm silicon process, synthesized and verified with post-synthesize simulation using EDA tools from Synopsys. A total of 32 processing cores are planned in our design, and the overall silicon area is estimated about 130 mm^2 in 2D mesh. The simulated working

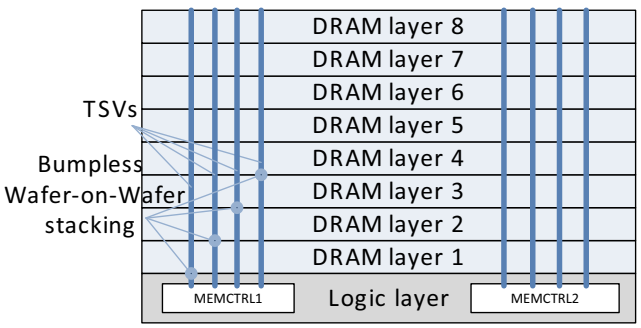


Figure 11 3D–stacked DRAM layers above single processing core.

frequency in our design is 333Mhz for 3D DRAM layers and DRAM controller, and 800Mhz for other function blocks in the logic layer. Simulation of short read mapping against whole human genome is impossible on RTL level of such a complicate design, therefore we make use of a FPGA board, terasIC DE5-NET equipped with 5SGXEA7N2F45C2N, which could verify one processing core with two DDR3 controllers function in real, and then we could scale the test results in order to compare with other platforms. The gate counts of each processing core is approximately 6 M, and the *Smith-Waterman* systolic array within each processing core consists of 1024 processing elements, which consumes about 68% of silicon area of a processing core.

BFAST acceleration was accomplished on FPGA platform by Olson et al. [9] on a system equipped with eight LX240T-2 Virtex-6 FPGA, evaluated an unknown dataset which consists of 50 million single end *reads*. Their reported experiment was 76 bp *reads* mapping to human chromosome 23, thus we have to scale their performance results in order to fit the case of whole human genome mapping. We assume an ideal FPGA platform that could fit all the required data of whole human genome mapping within onboard RAM, and the overall running time could be roughly estimated by Eq. 5:

$$T_{hg38} = T_{chr23} \times \left(Num\_READ_{hg38} \Big/ Num\_READ_{chr23} \right) \quad (5)$$

The different length of short *reads* would not affect much over the *seed* matching part, but run time of Smith-Waterman local alignment is correlated to sequence length. However Olson et al. didn't report the Smith-Waterman performance of their platform individually, thus the estimation given by Eq. 1 shall be better than it could perform in real.

GPU acceleration has been introduced for short read mapping problems, but the most recent skeleton work still lacks the support over whole human genome sequencing. Running on our workstation equipped with nVIDIA Geforce GTX 680, CUSHAW2-GPU v2.4.3 [11], which is the most recent implementation of short read mapping on GPU, reported error during loading the index dataset into onboard memory. Therefore we have to skip the comparison between our design and GPGPU, since an equivalent simulation is not possible without enough onboard memory.

5.2 Mapping Time

The run time between original BFAST, scaled FPGA platform, and estimation of our 3D–stacked architecture are shown in Table 1. Considering the extremely huge size of data files, e.g. 143GB index files, the time consumed on disk I/O are omitted for all the platforms. The time for data transaction between host and hardware acceleration platform are also omitted, though the time consumed on overall data preparation will be much more than computation without virtual ramdisk.

Table 1 Mapping time.

SRR867061	Single End		Paired End	
	Run time	Speed ratio	Run time	Speed ratio
BFAST	1778 m	1	3517 m	1
FPGA[a,b]	434.38 s	246×	-	-
3D VLSI 1[c]	2.37 s	45,012×	10.47 s	21,102×
3D VLSI 2[d]	64.49 s	1,654×	573.61 s	385×

[a] FPGA performance is scaled with Eq. 2

[b] FPGA platform consists of 8 FPGA chips, each with exclusive external memory interface

[c] 3D VLSI 1 performance is scaled from one processing core with 64 Smith-Waterman PEs on DE5

[d] 3D VLSI 2 performance is using different *seed* generation policy for more *seed* coverage

Results labelled with "3D VLSI 1" are generated by conformal with Original BFAST program, while "3D VLSI 2" is results with adjusted *seed* generation policy for higher sensitivity. The evaluation results show that the acceleration with "3D VLSI 1" has an amazingly 45,012 times speedup over original BFAST for single end *reads*, which also beat single FPGA for 1466 times. The acceleration over paired end *reads* is slightly lower but still as good as 21,102 times.

The DDR3 DRAM channels on DE5 are working on 667 MT/s. Since the DRAM access pattern is completely random, double date rate does little help for real bandwidth. The stacked DRAM layers gives single sub-processor the peak transfer rate of 64 × 2 × 333/8 = 5328 MB/s. This value is theoretically worse than that of DE5 board with two DDR3–667 channels giving 10.67GB/s, or the new WIDE I/O 2 [9] standard of 34GB/s. But the idea of extremely low latency and high bandwidth over random access shines: 3D–stacked SDR DRAM provides the best effort for low latency random memory access, while the sequential transfer rate is also sufficient for maximum *read* length of 8.

According to the recorded DRAM access traces, the effective read bandwidth on DRAM channel 0 of DE5 is approximately 450 MB/s, which consists of accesses to pointer table and original reference sequence. Accessing to DRAM channel 1 is less frequent; therefore DRAM channel 0 is the most critical system bottleneck.

According to simulated DRAM access traces of 3D SDRAM with 333Mhz, the random access is highly benefit from *Scatter/Gather* read policy, and the effective read bandwidth of DRAM channel 0 is about 1210 MB/s. Therefore the results "3D VLSI 1" are given by scaling of the FPGA results with a factor of (1210/450) × 32 linearly, as we don't have to consider data congestion with individual broadcast bus. The working frequency of processor layer is ignored here, since both 3D VLSI and stripped FPGA design is sufficient to process those data. There is no DRAM write operation during mapping, thus DRAM write is also ignored for this estimation.

Original BFAST program only use small suffix part of short *read* for *seed* generation, e.g. first 22 symbols of total 101 in length of each *read*. Definitely this would lead to miss mapping. The results of "3D VLSI 2" is scaled with the same method as "3D VLSI 1", but the *seed* generation is optimized for higher mapping sensitivity. The original BFAST application extracts the first 22 nucleotides of each *read* as *seed*, while in "3D VLSI 1" we try to pick a *seed* for every 22 symbols of each short *read* if possible, so that the *seed* coverage is much higher than original BFAST. This approach results very high sensitivity as described in section C.

5.3 Sensitivity

To measure mapping quality, the sensitivity is calculated by dividing the number of aligned short *reads* by the total number of short *reads* as (6):

$$\text{Sensitivity} = Num\_READ_{\text{mapped}} / Num\_READ_{\text{overall}} \quad (6)$$

Using the same attributes for *seed* generation and gap control, our design appears to be around the same sensitivity as original BFAST, shown as "3D VLSI 1" in Table 2. The optimized *seed* generation policy leads to much higher sensitivity while sacrificed some performance improvements.

5.4 Power Estimation

The logic layer of one sub-processor running at 800Mhz on 45-nm process consumes around 0.9 W given by post-synthesis simulation. This is achieved by using power gating on Smith-Waterman processing array, as it's idle by 62% of execution time. Therefore our example design would consume around 29 W energy on the logic layer with 130 mm^2 silicon areas.

The simulated memory traces depicted that the classical memory controller optimization methods does not apply to fully random accesses. With classical CACTI model [25], the estimated maximum power consumption for all the 8 DRAM layers is a bit less than 60 W.

In order to estimate accurate power consumption of 3D–stacked DRAM layers, the simulated DRAM access traces has been recorded and replayed in the DRAMSys tool [26]. Comparing with the theoretical maximum power consumption given by CACTI, estimated power consumption of 3D DRAM sub-system given by TLM models has been greatly reduced to 46 W in total, results from real transaction rate, *Scatter/Gather* read policy and scheduled power down.

This power density is significantly below the maximum value of 200 W/cm2 as defined by International Technology Roadmap for Semiconductors [27].

Table 2 Sensitivity.

SRR867061	Single end	Paired end	
	Sensitivity	Sensitivity	Confidence
BFAST	88.853%	90.773%	86.646%
3D VLSI 1	88.868%	90.811%	86.646%
3D VLSI 2	97.271%	97.963%	94.522%

6 Conclusion

We have described an approach to improve the performance of hash-index based short read mapping algorithm using a hardware acceleration platform.

We summarized the BFAST application with computation stages and data flow, and identified that 2-hit string matching and *Smith-Waterman* local sequence alignment consume a dominant portion of the overall execution time.

Then we introduced a customized processing core with dedicated DRAM channels, which is designed to facilitate 2-hit string matching and banded *Smith-Waterman* local sequence alignment. Parallelization of the two compute intensive stages has been well described, while both stages could benefit from exclusive low latency access to local DRAM channels. With customized dataset partitioning and task scheduling, the overlapped run time of these two major computation stages gives another factor of performance improvements over software implementation.

In our approach, a distributed memory interface with 3D–stacked architecture has been proposed to provide exclusive local access to each partitioned dataset. We introduced a *Scatter/Gather* read policy on DRAM controller to maximize the throughput of local random DRAM read access.

We present results from prototyping stripped down implementation of our design on a commodity FPGA, representing the performance of a single processing core of our architecture. Then we scale the results to give a plausible estimation of our design with customized many-core and 3D stacked architecture. Based on conservative area estimation and simulation results, we show that our design achieves the highest performance and greatest energy efficiency comparing with all other known accelerators.

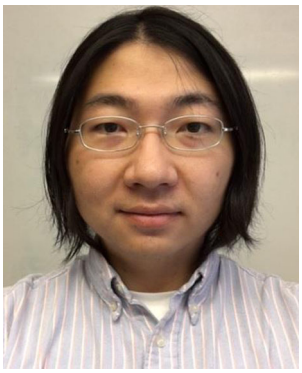
For sensitivity, the different indices and seeds combination could avoid most read errors, but the matching result can be further improved by extending the seed generation policy for higher coverage of short reads. This method would greatly impact the performance of software implementation, but suite well on our hardware acceleration platform.

Our platform could accelerate hash index short read mapping only at this moment, support of FM-index mapping is planned in the future.

Acknowledgements We are very grateful to Professor Lars Arvestad for providing many valuable suggestions.

References

1. Ferragina, P., & Manzini, G. (2000). Opportunistic data structures with applications. *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*. pp 390.
2. Smith, T. F., & Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology, 147*, 195–197.
3. Homer, N., Merriman, B., & Nelson, S. F. (2009). BFAST: an alignment tool for large scale genome resequencing. *PloS One, 4*(11), e7767.
4. Burrows, M., & Wheeler, D. J. (1994). A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation
5. Manber, U., & Myers, G. (1990). Suffix arrays: a new method for on-line string searches. First Annual ACM-SIAM Symposium on Discrete Algorithms. pp 319–327.
6. Langmead, B., Trapnell, C., Pop, M., & Salzberg, S. L. (2009). Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology, 10*, R25.
7. Li, H., & Durbin, R. (2009). Fast and accurate short read alignment with Burrows-Wheeler Transform. *Bioinformatics, 25*, 1754–1760.
8. illumina. illumina HiSeq 2500 ultra-high-throughput sequencing system. http://www.illumina.com/systems/hiseq_2500_1500.html. Accessed 31 Jan 2016.
9. Olson, C. B., Kim, M., Clauson, C., Kogon, B., Ebeling, C., Hauck, S., Ruzzo, W. L. (2012). Hardware acceleration of short read mapping field-programmable custom computing machines (FCCM). IEEE 20th annual international symposium on. pp 161–168.
10. Sogabe, Y. & Maruyama, T. (2013). An acceleration method of short read mapping using FPGA. Field-Programmable Technology (FPT). International Conference on. pp 350–353.
11. Liu, Y., & Schmidt, B. (2014). CUSHAW2-GPU: empowering faster gapped short-read alignment using GPU computing. *IEEE Design and Test of Computers, 31*(1), 31–39.
12. Roy Longbottom. RandMem Benchmark. http://www.roylongbottom.org.uk/randmem%20results.htm. Accessed 31 Jan 2016.
13. Intel®, Core™ i7 Processor. http://ark.intel.com/products/family/59143/. Accessed 31 Jan 2016.
14. JEDEC. DDR3 SDRAM Standard, JESD79-3F. (2012). https://www.jedec.org/standards-documents/docs/jesd-79-3d. Accessed 31 Jan 2016.
15. Abts, D., Enright Jerger, N. D., Kim, J., Gibson, D., and Lipasti, M. H. (2009). Achieving predictable performance through better memory controller placement in many-core CMPs. *In Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA '09)*, *37*(3), 451–461.
16. Loi, I., & Benini, L. (2010). An efficient distributed memory interface for many-core platform with 3D stacked DRAM. *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp 99–104.
17. Weis, C., Loi, I., Benini, L., & Wehn, N. (2013). Exploration and optimization of 3-D integrated DRAM subsystems. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on, 32*(4), 597–610.
18. Sandhu, G. (2012). (Micron Technology, Inc.). DRAM scaling & bandwidth challenges in Proc. NSF WETI.
19. Hybrid Memory Cube Consortium. Hybrid Memory Cube. http://www.hybridmemorycube.org. Accessed 31 Jan 2016.
20. Ibarra, O., & Palis, M. (Jul 1987). VLSI algorithms for solving recurrence equations and applications. *Acoustics, Speech and Signal Processing, IEEE Transactions on, 35*(7), 1046–1064.
21. Chao, K. M., Pearson, W. R., & Miller, W. (1992). Aligning two sequences within a specified diagonal band. *Computer Applications in the Biosciences, 8*(5), 481–487.
22. JEDEC. Wide I/O 2 (WideIO2), JESD229-2. (2014).https://www.jedec.org/standards-documents/docs/jesd229-2. Accessed 31 Jan 2016.
23. International Genome Sample Resource. 1000 Genomes Project. http://www.1000genomes.org. Accessed 31 Jan 2016.
24. Miga, K. H., Newton, Y., Jain, M., Altemose, N., Willard, H. F., & Kent, W. J. (2014). Centromere reference models for human chromosomes X and Y satellite arrays. *Genome Research, 24*(4), 697–707.
25. International Technology Roadmap for Semiconductors. http://www.itrs.net. Accessed 31 Jan 2016.
26. HP Labs. CACTI. http://www.hpl.hp.com/research/cacti/. Accessed 31 Jan 2016.
27. Jung, M., Weis, C., & Wehn, N. (2005). DRAMSys: a flexible DRAM subsystem design space exploration framework. IPSJ Transactions on System LSI Design Methodology (T-SLDM). October, 2015.



Pei Liu is pursuing his Ph.D. in the Department of Electronic Systems, School of ICT, KTH, Kista, Sweden. His research interests are coarse-grained reconfigurable architecture for accelerating bioinformatics algorithms and applications, especially with the focus on the biological sequence analysis problems.

Ahmed Hemani is a Professor in Electronic Systems Design at School of ICT, KTH, Kista, Sweden. His current areas of research interests are massively parallel architectures and structured VLSI design methods and their applications to scientific computing and autonomous embedded systems inspired by brain. He has proposed a structured VLSI design method called SiLago (Silicon Large Grain Object) as an alternative to the standard cell based design flow. In past, he has contributed to high-level synthesis – his doctoral thesis was the basis for the first high-level synthesis product introduced by Cadence called visual architect. He has also pioneered the Networks-on-chip concept and has contributed to clocking and low power architectures and design methods. He has extensively worked in industry including National Semiconductors, ABB, Ericsson, Philips Semiconductors, Newlogic. He has also been part of three start-ups and is a senior member of IEEE.

Kolin Paul is an Associate Professor in the Department of Computer Science and Engineering at IIT Delhi India. He received his B.E. degree in Electronics and Telecommunication Engineering from NIT Silchar in 1992 and Ph.D. in Computer Science in 2002 from BE College (DU), Shibpore. During 2002-3 he did his post doctoral studies at Colorado State University, Fort Collins, USA. He has previously worked at IBM Software Labs. His last appointment was as a Lecturer in the Department of Computer Science at the University of Bristol, UK. He has also held a Visiting Position at KTH, Stockholm. His research interests are in understanding high performance architectures and compilation systems. In particular he works in the area of Adaptive/Reconfigurable Computing trying to understand its use and implications in embedded systems. He is also involved in the design of systems for affordable healthcare.

Christian Weis received the Ph.D. degree in electrical engineering from the University of Kaiserslautern, Kaiserslautern, Germany, in 2014. From 1996 to 1998, he was with Mitsubishi Semiconductor Europe, Germany, where he was engaged in the development of microcontrollers. From 1998 to 2009, he was with Siemens Semiconductor, Infineon Technologies AG and Qimonda AG, Munich, Germany, in DRAM design. During this time frame, he was involved in DRAM design for graphics and commodity DRAM products. In 2006, he was a Design Team Leader for the 1Gb DDR3 DRAM, the first DDR3 volume product at infineon/Qimonda. Since 2009, he has been with the Microelectronic System Design Research Group, University of Kaiserslautern, Kaiserslautern, Germany. He holds several patents related to DRAMs and DRAM design. His current research interests include 3D integrated DRAMs, DRAM controller design, and heterogeneous memories and MPSoCs.

Matthias Jung received the Diploma degree in electrical engineering from the University of Kaiserslautern, Kaiserslautern, Germany in 2011. Since this time, he is pursuing his Ph.D. in the Microelectronic Systems Design Research Group of the University of Kaiserslautern. His research interests are SystemC based virtual prototypes, especially with the focus on the modelling of memory systems, memory controllers and DRAM architecture.

Norbert Wehn holds the chair for Microelectronic System Design in the department of Electrical Engineering and Information Technology. He is also the Vice President at the University of Kaiserslautern and has more than 300 publications in various fields of microelectronic system design. He holds several patents and has founded two Start-Ups. He served among others as program chair for DATE 2003, general chair for DATE 2005 and general Co-Chair at FPL 2014. He is associate editor of various journals and member of several scientific advisory boards. His special research interests are VLSI-architectures for mobile communication, forward error correction techniques, advanced SoC architectures and methodologies, 3D integration, reliability issues in SoC and high performance computing on embedded devices.